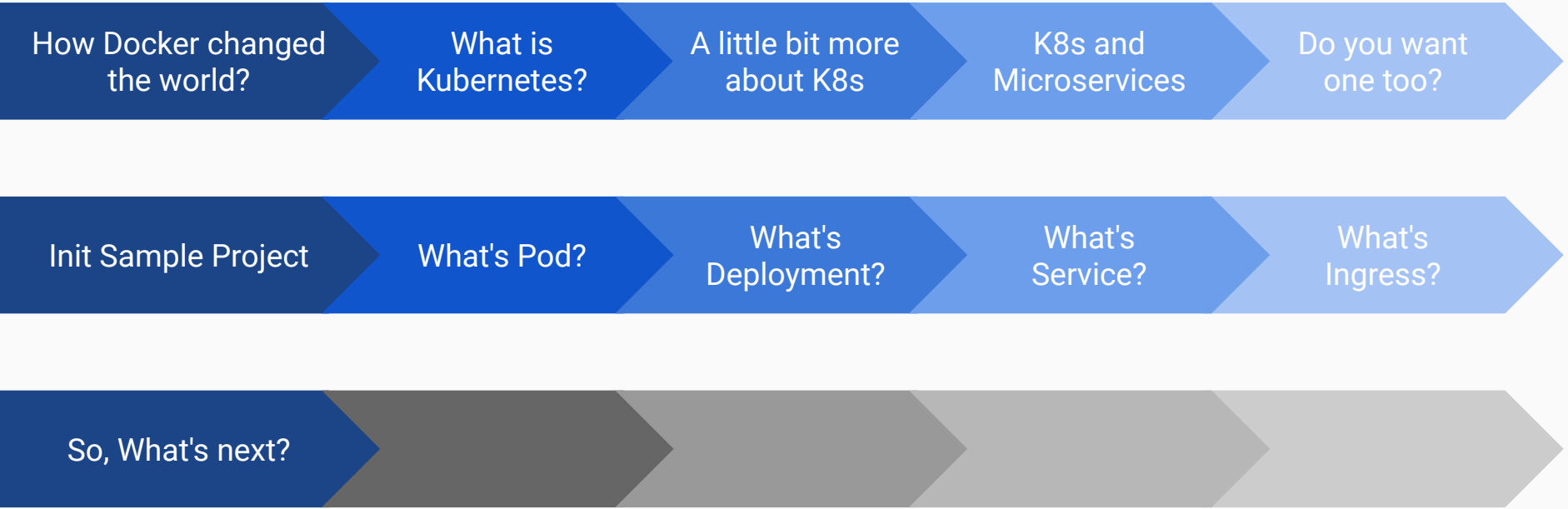
Kubernetes

Podro Workshop

In this workshop



How Docker changed the world?

What is Docker?

In 2013, Docker introduced what would become the industry standard for containers. Containers are a standardized unit of software that allows developers to isolate their app from its environment, solving the “it works on my machine” headache. For millions of developers today, Docker is the de facto standard to build and share containerized apps - from desktop, to the cloud. We are building on our unique connected experience from code to cloud for developers and developer teams.

Benefits

1	Portability	Docker has its own advantages when it comes to Portability. Besides that, we no longer require a virtual machine spun up for each and every app. Why? Because if I run a CoreOS host server and have a guest Container-based off of Ubuntu, the Container has the parts that make Ubuntu different from CoreOS. While a Virtual Machine is a whole other guest computer running on top of your host computer (sitting on top of a layer of Virtualization), Docker is an isolated portion of the host computer, sharing the host kernel (OS) and even its binaries/libraries if appropriate.
2	Shared Resources	Docker uses shared Kernel which means that they are much more efficient than hypervisors in terms of system resources. Containers are lightweight which means that there are several VMs that can each run its own application with its own operating system. But there can be thousands of containers in the same server that shares the Kernel's operating system to execute the work. This, in turn, means you can leave behind the useless 99.9% VM junk, leaving you with a small, neat capsule containing your application.
3	Easy to launch	Currently, in order to run on a remote cloud server, one has to log in to cloud server (install Docker) and push the image (that your devices have access to) to Docker Registry after which you pull that image down to the cloud server. Only after this when the cloud server is ready to go, can you launch the container.
4	Zero Downtime Deployment	In terms of Deployment, it is always a race about who can deploy often and fast, be fully automatic, accomplish zero-downtime, have the ability to rollback, provide constant reliability across environments, be able to scale effortlessly and create self-healing systems able to recuperate from failures. This means that you can have DevOps Continuous Integration (CI), Continuous Delivery / Deployment (CD) and leverage Docker ecosystem to deliver often and fast, be fully automatic, accomplish zero-downtime, have the ability to rollback software products.

More Benefits...

- Agile application creation and deployment: increased ease and efficiency of container image creation compared to VM image use.
- Continuous development, integration, and deployment: provides for reliable and frequent container image build and deployment with quick and easy rollbacks (due to image immutability).
- Dev and Ops separation of concerns: create application container images at build/release time rather than deployment time, thereby decoupling applications from infrastructure.
- Observability not only surfaces OS-level information and metrics, but also application health and other signals.
- Environmental consistency across development, testing, and production: Runs the same on a laptop as it does in the cloud.
- Cloud and OS distribution portability: Runs on Ubuntu, RHEL, CoreOS, on-premises, on major public clouds, and anywhere else.
- Application-centric management: Raises the level of abstraction from running an OS on virtual hardware to running an application on an OS using logical resources.
- Loosely coupled, distributed, elastic, liberated micro-services: applications are broken into smaller, independent pieces and can be deployed and managed dynamically – not a monolithic stack running on one big single-purpose machine.
- Resource isolation: predictable application performance.
- Resource utilization: high efficiency and density.

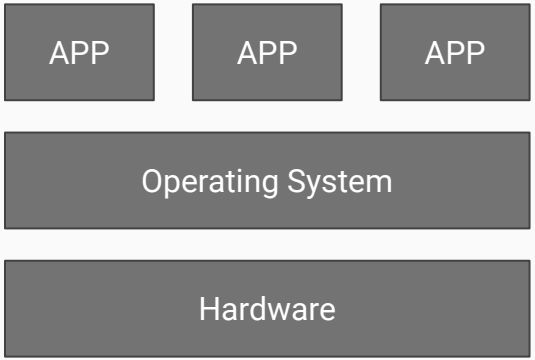
Sample Docker Run Command

```
docker run -p 8080:80 -d mendhak/http-https-echo
```

What Is Kubernetes?

Traditional Deployment

Early on, organizations ran applications on physical servers. There was no way to define resource boundaries for applications in a physical server, and this caused resource allocation issues. For example, if multiple applications run on a physical server, there can be instances where one application would take up most of the resources, and as a result, the other applications would underperform. A solution for this would be to run each application on a different physical server. But this did not scale as resources were underutilized, and it was expensive for organizations to maintain many physical servers.



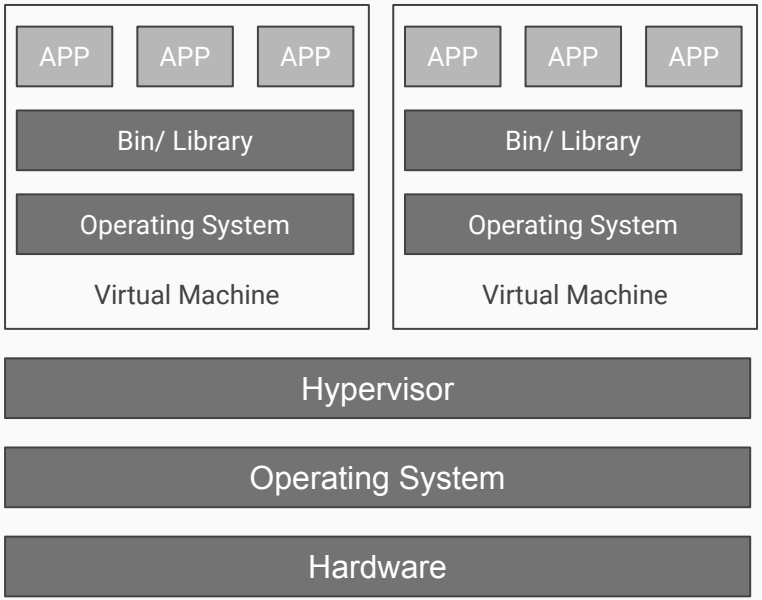
read more: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Virtualized Deployment

As a solution, virtualization was introduced. It allows you to run multiple Virtual Machines (VMs) on a single physical server's CPU. Virtualization allows applications to be isolated between VMs and provides a level of security as the information of one application cannot be freely accessed by another application.

Virtualization allows better utilization of resources in a physical server and allows better scalability because an application can be added or updated easily, reduces hardware costs, and much more. With virtualization you can present a set of physical resources as a cluster of disposable virtual machines.

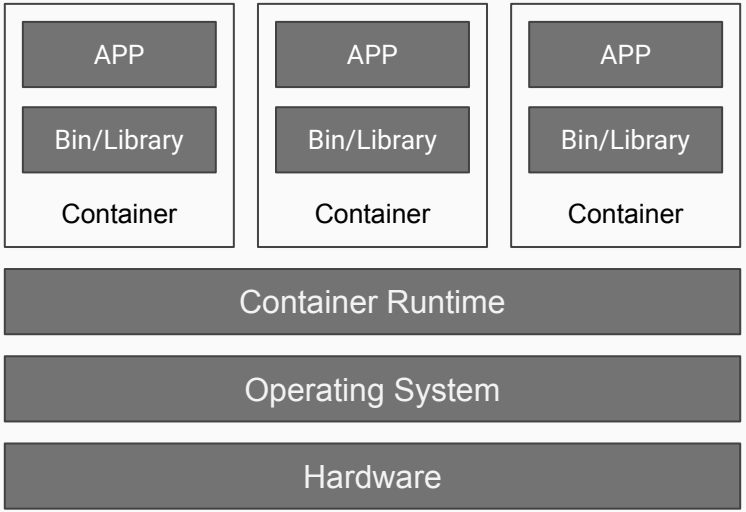
Each VM is a full machine running all the components, including its own operating system, on top of the virtualized hardware.



read more: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Container Deployment

Containers are similar to VMs, but they have relaxed isolation properties to share the Operating System (OS) among the applications. Therefore, containers are considered lightweight. Similar to a VM, a container has its own filesystem, CPU, memory, process space, and more. As they are decoupled from the underlying infrastructure, they are portable across clouds and OS distributions.



read more: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It has a large, rapidly growing ecosystem. Kubernetes services, support, and tools are widely available.

read more: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

A Little bit More About K8s

What K8s Provides?

- Service discovery and load balancing

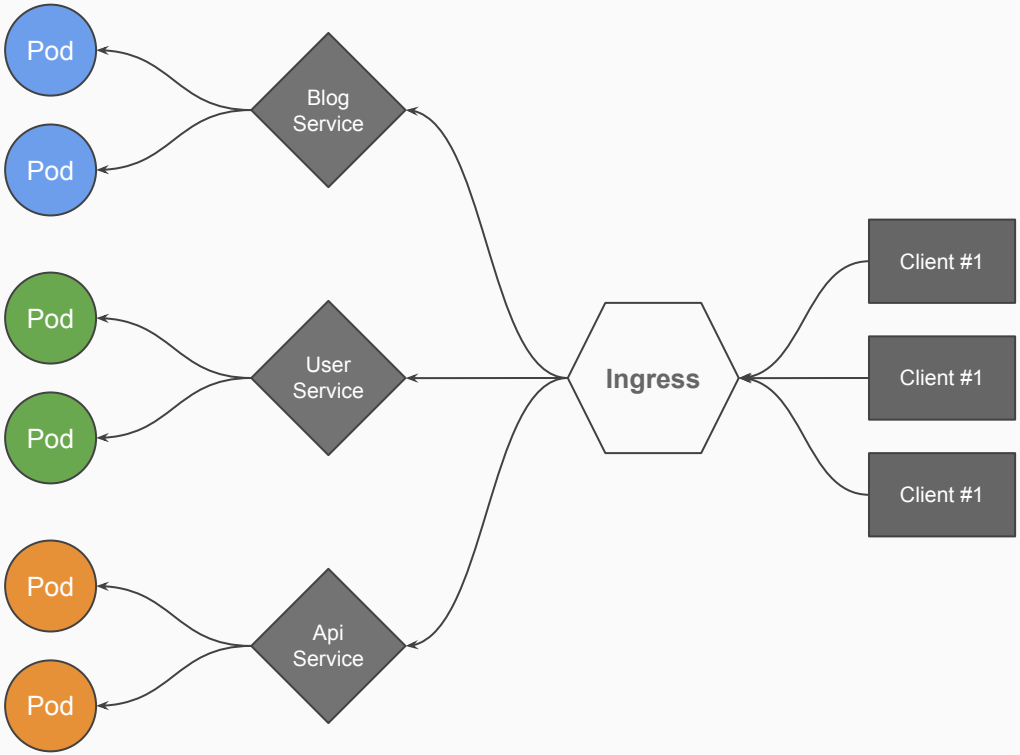
- Storage orchestration

- Automated rollouts and rollbacks

- Automatic bin packing

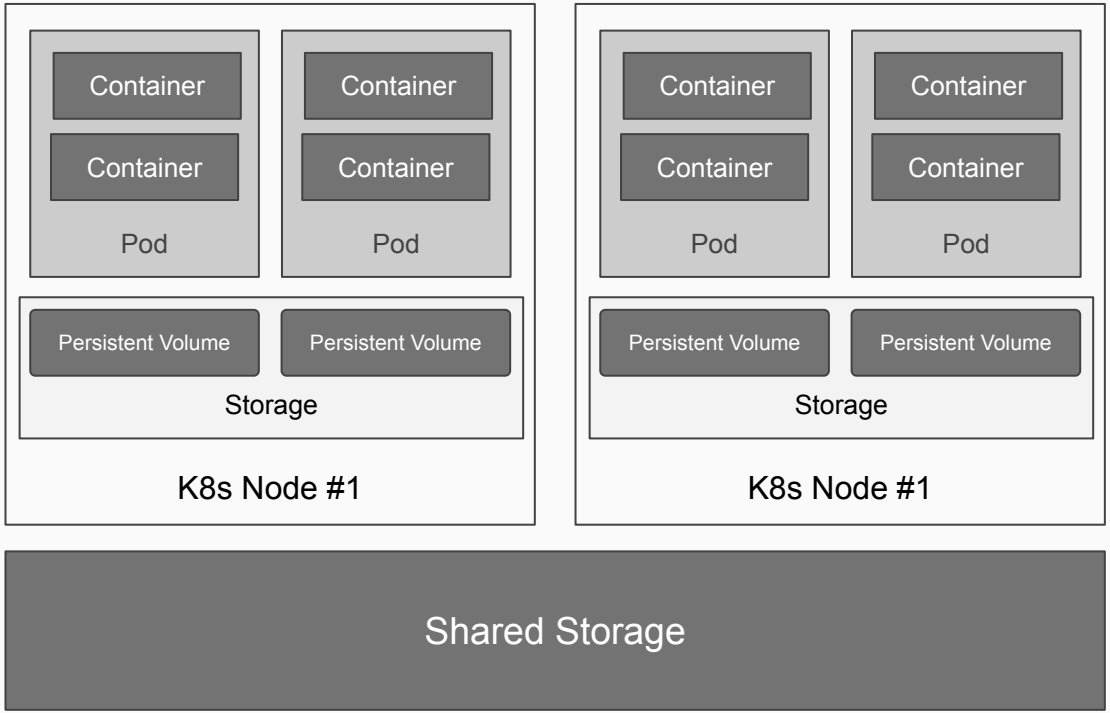
- Self-healing

- Secret and configuration management



What K8s Provides?

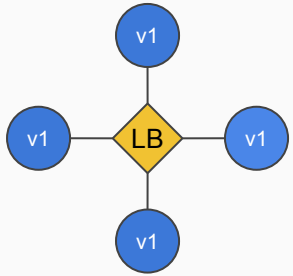
- Service discovery and load balancing
- **Storage orchestration**
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management



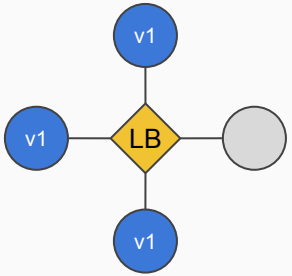
What K8s Provides?

- Service discovery and load balancing
- Storage orchestration
- **Automated rollouts and rollbacks**
- Automatic bin packing
- Self-healing
- Secret and configuration management

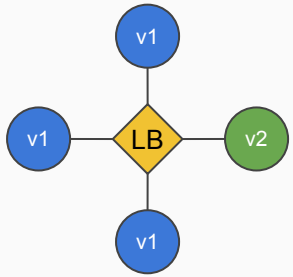
1.



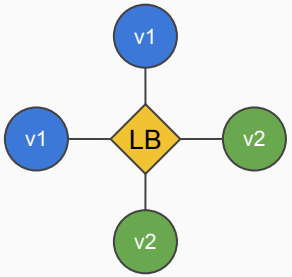
2.



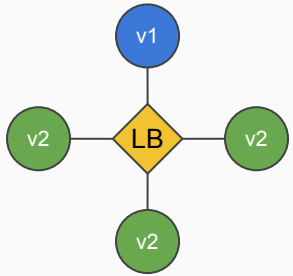
3.



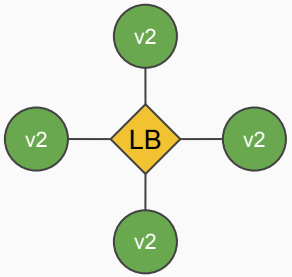
4.



5.

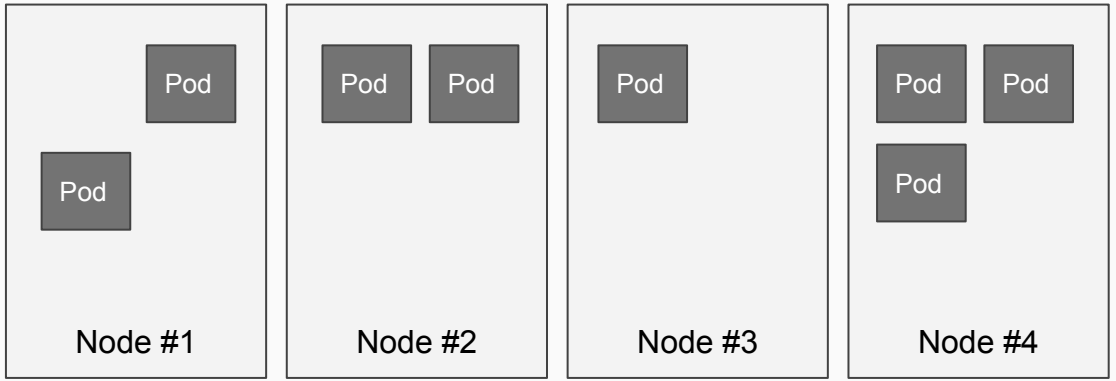


6.



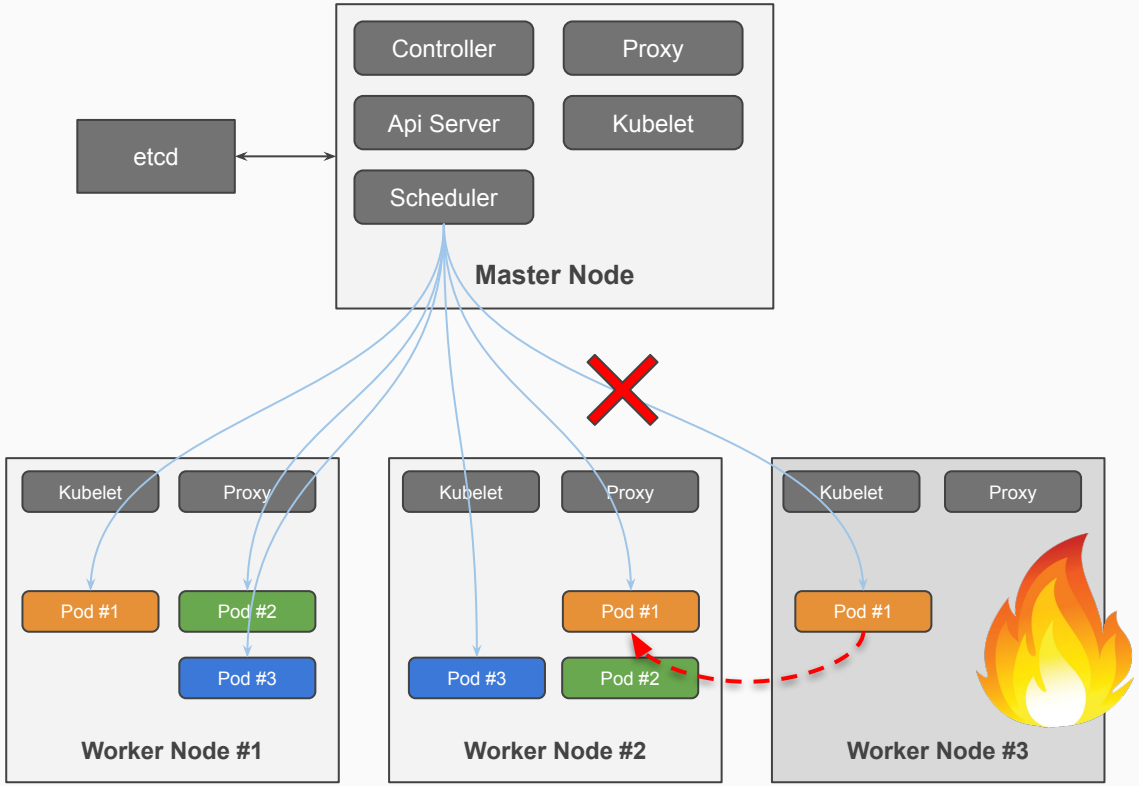
What K8s Provides?

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- **Automatic bin packing**
- Self-healing
- Secret and configuration management



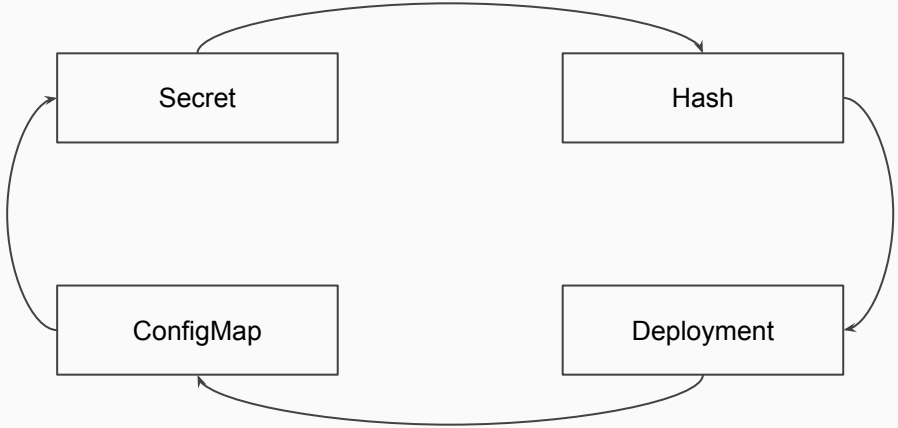
What K8s Provides?

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- **Self-healing**
- Secret and configuration management



What K8s Provides?

- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- **Secret and configuration management**



K8s and Microservices

Kubernetes can do...

- Assist very well in running automated CI/CD.
- Managing disparate isolated settings, resources, storage distributions.
- Deployments and rollbacks with automatic scheduling, service detection, and load balancing.
- Maintaining resilience and fault tolerance becomes easier and effective.
- Easy to deal with app configurations and executing centralized logging systems, metrics gathering and tracing.
- Executing stateful services, scheduled jobs and batch jobs with ease and efficiency.
- Provides a lot of innovative time to try hands-on newer things like auto replication, auto-scaling, etc.

Do You Want One Too?

Access to Minikube

As a playground:

https://kubernetes.io/docs/tutorials/hello-minikube/

As a development environment:

https://kubernetes.io/docs/tasks/tools/install-minikube/

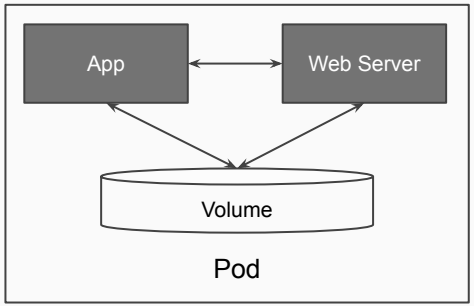
What’s Pod?

Definition

A Pod is a group of one or more containers, with shared storage/network, and a specification for how to run the containers.

Pod's common statuses:

Pending	One or more of the Container images has not been created.
Running	All of the Containers have been created. At least one Container is still running.
Succeeded	All Containers in the Pod have terminated in success.
Failed	All Containers in the Pod have terminated, and at least one Container has terminated in failure.
Unknown	For some reason the state of the Pod could not be obtained.

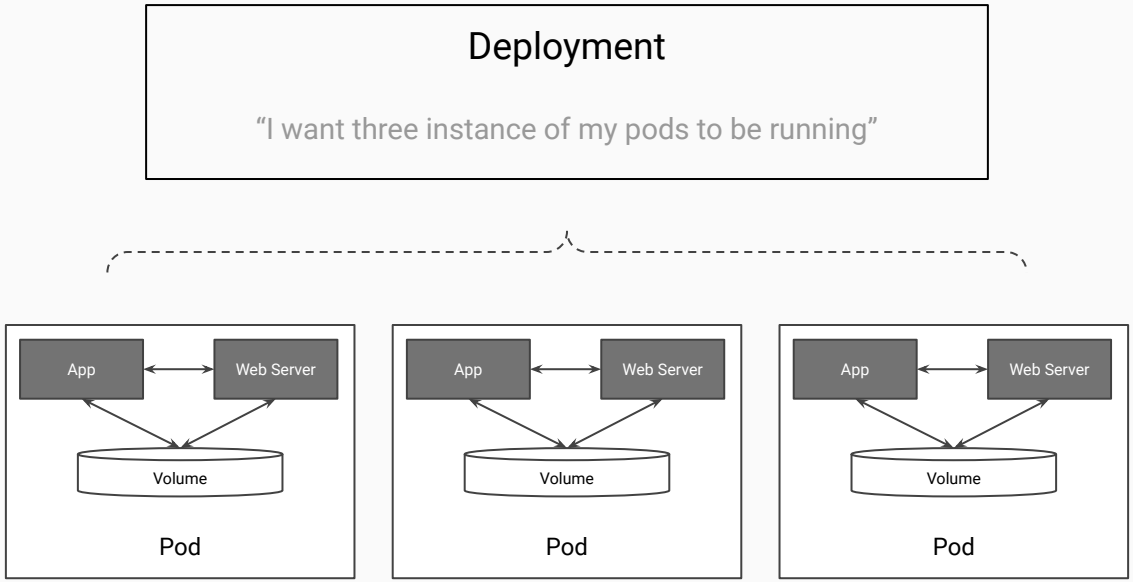


read more: https://kubernetes.io/docs/tasks/configure-pod-container/
https://kubernetes.io/docs/tasks/configure-pod-container/configure-pod-initialization/

What’s Deployment?

Definition

Deployments represent a set of multiple, identical Pods with no unique identities. A Deployment runs multiple replicas of your application and automatically replaces any instances that fail or become unresponsive. ... Deployments are managed by the Kubernetes Deployment controller.

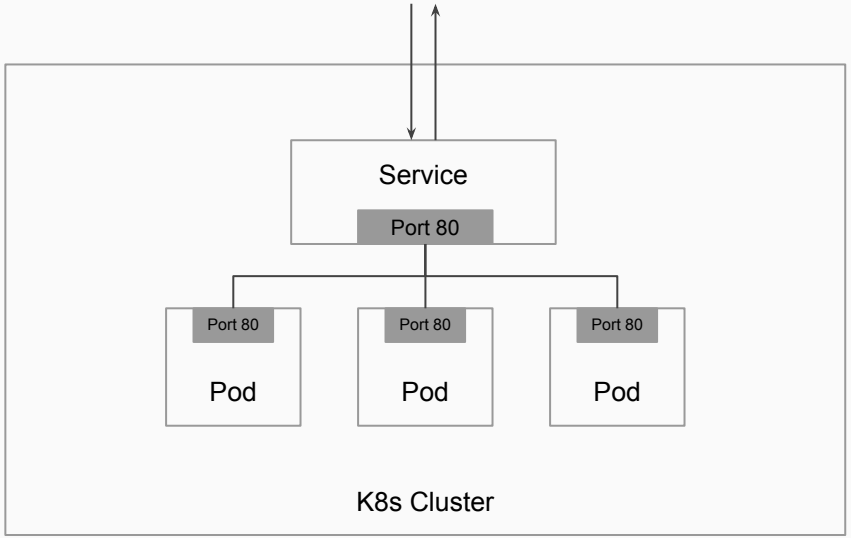


read more: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

What’s Service?

Definition

In Kubernetes, a Service is an abstraction which defines a logical set of Pods and a policy by which to access them. The set of Pods targeted by a Service is usually determined by a selector.

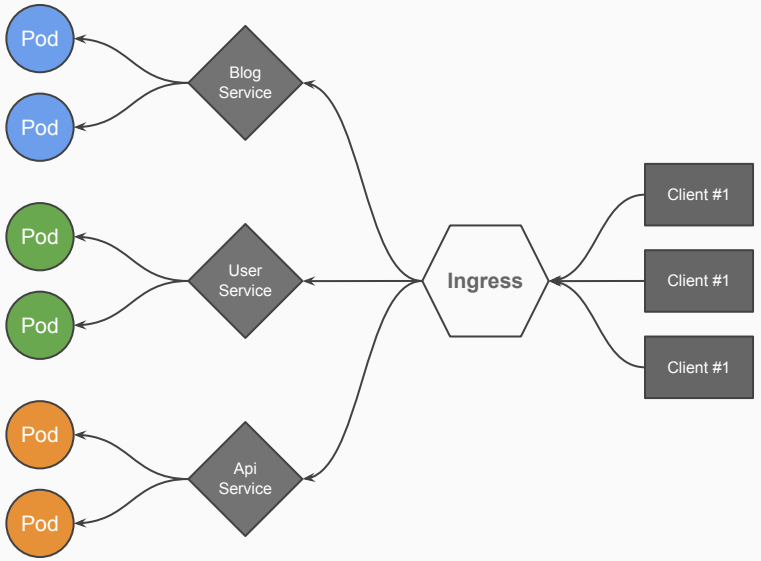


What’s Ingress?

Definition

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

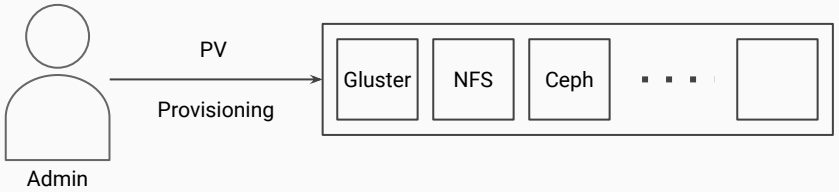
An Ingress may be configured to give Services externally-reachable URLs, load balance traffic, terminate SSL / TLS, and offer name based virtual hosting. An Ingress controller is responsible for fulfilling the Ingress, usually with a load balancer, though it may also configure your edge router or additional frontends to help handle the traffic.



What’s PV and PVC?

Definition

A PersistentVolume (PV) is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It is a resource in the cluster just like a node is a cluster resource. PVs are volume plugins like Volumes, but have a lifecycle independent of any individual Pod that uses the PV. This API object captures the details of the implementation of the storage, be that NFS, iSCSI, or a cloud-provider-specific storage system.



Definition

A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see AccessModes).